INOVA PRODEMGE
1ª semana de inovação

JVM

ANALISANDO DESEMPENHO DE VM'S 8, 9, 10 E 11 COM APLICAÇÕES JAVA

INOVA PRODEMGE
1ª semana de inovação

AGENDA

- JVM
- Garbage Collector
- Runtime
- Projetos
- Prática

INOVA PRODEMGE
1ª semana de inovação

$ whoami

- Leonardo Savio
  - Trabalha na Prodemge como lider técnico core criptográfico da AC de Nível 1
  - Graduado em Sistemas de Informação e pós graduado em Desenvolvimento de Sistemas Web
  - Trabalha com Java por 15+ anos
    - Desktop(JavaFX), Web(EE), Mobile(GluonHQ) e Embedded(ARM JavaME/Embedded)
  - Trabalhou com C por 5+ anos
    - Desenvolvimento de kernel(uClinux/FreeRTOS baremetal ARM), drivers e sistemas embarcados(multithread).

INOVA PRODEMGE
1ª semana de inovação

JVM

- Nessa apresentação apresentaremos algumas evoluções do Java 9, 10 e 11 em relação a GC, Runtime e realizaremos alguns testes com aplicações Java para análise de desempenho.

INOVA PRODEMGE
1ª semana de inovação

JAVA 9

App	Library	Library
*.class	*.jar	*.jar

Java Class Library	*.jmod

Java Virtual Machine

GC – Runtime – Compiler – libjvm.so

INOVA PRODEMGE
1ª semana de inovação

JVM - JAVA VIRTUAL MACHINE

- Compoentes
  - GC Garbage Collector
    - Responsável por alocar memória e liberar a memória que não está mais em uso.
  - Runtime
    - Responsável por carregar as classes, interagir com Sistema operacional, etc.
  - Compiler
    - Responsável por compilar Java Bytecode em Código de máquina altamente eficiente.

INOVA PRODEMGE
1ª semana de inovação

JVM – GARBAGE COLLECTOR

- 1. Encontrar os abjetos em uso

- 2. Compactar os objetos em uso

JVM – GARBAGE COLLECTOR

- A compactação pode levar um grande tempo quando:
  - O heap estiver muito cheio
  - O heap contiver bilhões de objetos em uso
- A aplicação Java fica parada durante a compactação
- A aplicação pode ficar parada por muitos segundos

JVM – GARBAGE COLLECTOR

- G1 – Garbage Collector
  - G1 é o novo algoritmo gerenciador de memória padrão desde JDK 9
    - Suportado desde 7u4
  - Objetivo: taxa de transferência e baixa latencia.
  - A pausa padrão para o G1 é de 200 milisegundos
    - Pausa maior > mais transferencia, maior latencia
    - Pausa menor > menos transferencia, menor latencia

INOVA PRODEMGE
1ª semana de inovação

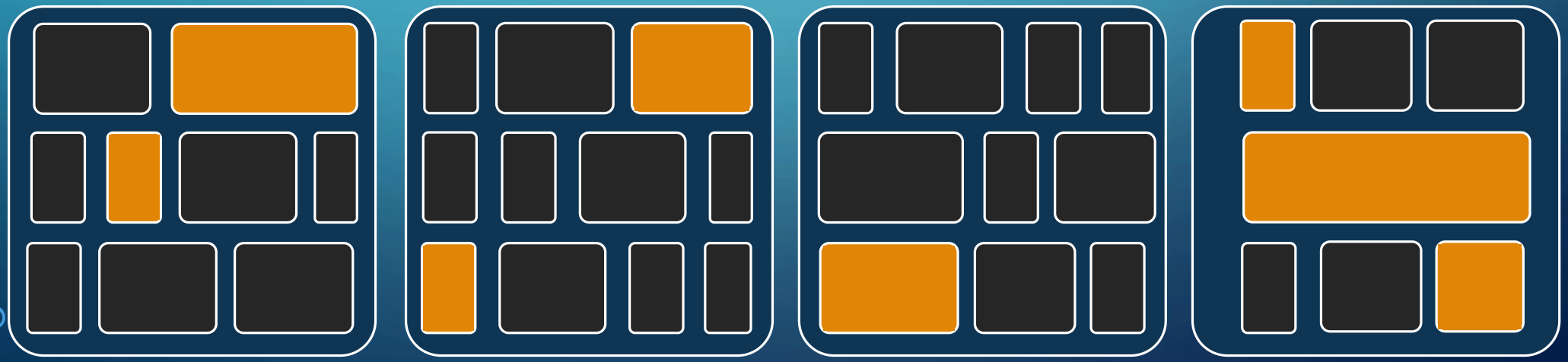
JVM – GARBAGE COLLECTOR

- G1 – Garbage Collector
  - G1 divide o heap em multiplas regioes
  - Encontra todos os objetos em uso concorrentemente
    - A aplicação Java não para

INOVA PRODEMGE
1ª semana de inovação

JVM – GARBAGE COLLECTOR

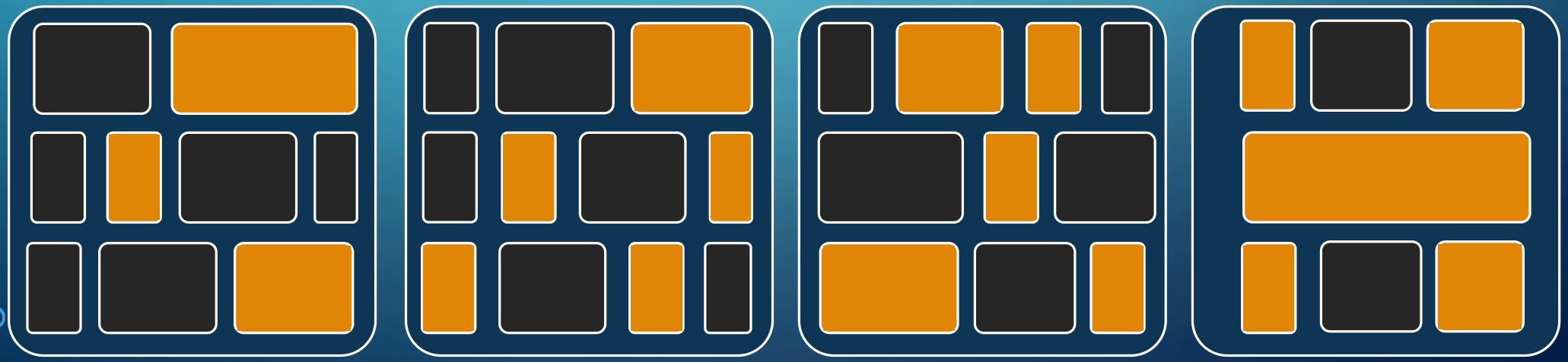
- G1 – Garbage Collector
  - G1 divide o heap em multiplas regioes
  - Encontra todos os objetos em uso concorrentemente
    - A aplicação Java não para



INOVA PRODEMGE
1ª semana de inovação

JVM – GARBAGE COLLECTOR

- G1 – Garbage Collector
  - G1 divide o heap em multiplas regioes
  - Encontra todos os objetos em uso concorrentemente
    - A aplicação Java não para
  - Mantém o controle dos ponteiros entre as regiões



JVM – GARBAGE COLLECTOR

- G1 – Garbage Collector
  - G1 divide o heap em múltiplas regiões
  - Encontra todos os objetos em uso concorrentemente
    - A aplicação Java não para
  - Mantém o controle dos ponteiros entre as regiões
  - G1 pode, portanto, coletar algumas regiões de cada vez
    - Menos objetos para coletar > menores pausas
    - Mais informações > maior controle das pausas

JVM – GARBAGE COLLECTOR

- G1 – Garbage Collector
  - O que acontece se o heap enche (full) antes do G1 encontrar todos os objetos ativos?
    - Voltamos a um algoritmo secundário: full collection
  - Esse algoritmo costuma ser uma thread única
  - Pode levar um tempo grande em heaps muito grandes.
  - Isto foi resolvido no JDK 10

JVM – GARBAGE COLLECTOR

INOVA PRODEMGE
1ª semana de inovação

- G1 – Garbage Collector
  - JEP 307: Parallel Full GC para G1
    - Consiste na utilização de threads paralelas.
  - Algoritmo “mark-sweep-compact” (marcar-varrer-compactar) em threads paralelas
  - Melhora significante nos piores casos de pausa do G1
  - Ativo por default
  - Disponível no JDK 10

JVM – GARBAGE COLLECTOR

- G1 – Garbage Collector
  - O G1 no JDK10 também obteve melhorias na varredura do heap
  - Pausas mais curtas
  - Menos uso concorrente de CPU
  - Ativo por default

JVM – RUNTIME

- Runtime
  - É responsável por várias partes
    - Carregar as classes Java
    - Interagir com o Sistema Operacional
    - Log
    - Diagnósticos
    - Serviços
    - Concorrência primitiva (threads, monitores, etc)

JVM – RUNTIME

- Carregando classes Java
  - As classes são desenvolvidas em arquivos texto exemplo:
    - App.java
  - Ao serem compiladas (javac) são gerados arquivos binários, exemplo:
    - App.class
  - Quando carregadas em memória (DefineClass()) se tornam objetos, exemplo:
    - Klass*

INOVA PRODEMGE
1ª semana de inovação

JVM – RUNTIME

- Carregando classes Java
  - Carregar uma classe Java de um arquivo .class demanda tempo
    - Para carregar .class do disco
    - Para verificar o .class
    - Para criar uma representação na JVM
    - Para resolver todas as referencias
  - Quantas vezes você reiniciar a mesma aplicação Java
    - Todo este trabalho será repetido a cada vez que a JVM inicializar
  - Isto pode ser otimizado?

JVM – RUNTIME

- Carregando classes Java
  - JDK 1.5 foi disponibilizado com o compartilhamento de dados de classes(CDS)
  - CDS (Class-Data Sharing) armazena dados já carregados e verificados de classes em um arquivo que pode ser compartilhado por múltiplas JVMs
  - Limitações
    - Somente para Serial Garbage Collector
    - Funciona somente com classes “system”
    - Somente client JVMs

JVM – RUNTIME

- Carregando classes Java
  - JDK 9 disponibilizado com Class-Data Sharing aprimorado
    - Funciona com G1, Parallel e Serial garbage collectors
    - Funciona com ambas client e server JVMs
  - Limitações
    - Continua aplicável apenas a classes “system”

JVM – RUNTIME

- Carregando classes Java
  - JDK 9 também foi disponibilizado com o Application Class-Data Sharing (AppCDS)
    - Habilita que classes da aplicação sejam armazenadas no arquivo compartilhado. Exemplo: Pessoa.class
  - Funcionalidade Comercial
  - Disponível apenas no JDK 9 Oracle

JVM – RUNTIME

- Carregando classes Java
  - AppCDS se tornou open source no JDK 10
  - JEP310 – Application Class-Data Sharing
  - Disponível nos builds Open JDK
  - Resulta em melhores tempos de inicialização (startup)

JVM – RUNTIME

- Carregando classes Java
  - Encontrar as classes para compartilhar:

    $ java -Xshare:off -XX:+UseAppCDS -XX:DumpLoadedClassList=app.lst -cp lib.jar App
  - Criar o arquivo compartilhado:

    $ java -Xshare:dump -XX:+UseAppCDS -XX:SharedClassListFile=app.lst -XX:SharedArchiveFile=app.jsa -cp lib.jar
  - Usar o arquivo compartilhado:

    $ java -Xshare:auto -XX:+UseAppCDS -XX:SharedArchiveFile=app.jsa -cp lib.jar App

JVM – RUNTIME

- Interagindo com o Sistema Operacional
  - O Runtime utiliza o Sistema operacional para:
    - Threads
    - Alocação de memória
    - Sockets
  - Em um mundo de containers é importante usar as APIs corretas
    - Não chamar a CPU diretamente
    - Estar cientes dos namespaces Linux

JVM – RUNTIME

- Interagindo com o Sistema Operacional
  - A JVM no JDK 10 usa as interfaces corretas do SO
  - Usa cggroups para encontrar os limites de memória
    - O tamanho padrão do heap não precisará ser tão grande
  - Usa cpuset para limitar o número de cores
    - O número padrão de threads do GC e do compilador estarão corretos
  - Usa o “pid namespace” para conectar a API
    - Ferramentas do JDK como o jmap consegue conectar numa JVM executando em um container

JVM – RUNTIME

- Interagindo com o Sistema Operacional
  - Novas e mais intuitivas fags para definir o tamanho do heap

    -XX:MinRAMPercentage

    -XX:InitialRAMPercentage

    -XX:MaxRAMPercentage
  - Funciona bem com containers
    - Não é mais necessário redefinir as flags quando a quantidade de memória do container for modificada

JVM – RUNTIME

- Interagindo com o Sistema Operacional
  - Novas e mais intuitivas fags para definir o tamanho do heap

    -XX:MinRAMPercentage

    -XX:InitialRAMPercentage

    -XX:MaxRAMPercentage
  - Funciona bem com containers
    - Não é mais necessário redefinir as flags quando a quantidade de memória do container for modificada

JVM – PROJETOS

- Alguns projetos:
  - ZGC
  - Portola
  - Valhalla
  - Loom
  - Panama
- Acesse http://openjdk.java.net para mais projetos e informações

JVM – PROJETOS

- Projeto ZGC
  - ZGC é um novo e escalável Garbage Collector de baixa latência
    - Tempo de pausas de 10 ms
    - Os tempos de pausa não aumentam com o tamanho do heap
    - Projetado para heaps de poucos gigabytes a multiplos terabytes
  - Procura todos os objetos ativos concorrentemente
  - Compacta os objetos ativos concorrentemente
  - Processa as referencias concorrentemente
  - http://jdk.java.net/zgc

JVM – PROJETOS

- Projeto Portola
  - É um port do OpenJDK para Aplpine Linux
  - Usa a musl C Standar Library (lightweight, rápida, simples, gratuita …)
  - O tamanho do Alpine Linux quando usado em uma imagem Docker: 4MB
    - Comparado ao Ubuntu: 120MB
  - Para gerar um build reduzido (small) da JRE Java Runtime Environment
    - O build Java básico para Docker Alpine Linux: 40MB
  - http://jdk.java.net/11

JVM – PROJETOS

- Projeto Portola
  - É um port do OpenJDK para Aplpine Linux
  - Usa a musl C Standar Library (lightweight, rápida, simples, gratuita …)
  - O tamanho do Alpine Linux quando usado em uma imagem Docker: 4MB
    - Comparado ao Ubuntu: 120MB
  - Para gerar um build reduzido (small) da JRE Java Runtime Environment
    - O build Java básico para Docker Alpine Linux: 40MB
  - http://jdk.java.net/11

JVM – PRÁTICA

- Executar aplicação utilizando diferentes builds de VMs Java e verificar a diferença de desempenho.

JVM – FIM

Obrigado

leonardo.ferreira@prodemge.gov.br